**1. Dinamički polimorfizam u C-u**

Uspostavite vezu s terminologijom iz objektno orijentiranih jezika. Koji elementi vašeg rješenja bi korespondirali s podatkovnim članovima objekta, metodama, virtualnim metodama, konstruktorima, te virtualnim tablicama?

objekt 🡪 struktura Animal (alocirana); malloc; instanciranje objekta: createDog, createCat

metode 🡪 animalPrintGreeting, animalPrintMenu

virtualne metode 🡪 dogGreet, dogMenu, catGreet, catMenu

konstruktori -> constructDog, constructCat

virtualne tablice (tj. tablice virtualnih funkcija) 🡪 dogTable, catTable

**3. Memorijska cijena dinamičkog polimorfizma**

Ispitajte memorijske zahtjeve objekata dvaju tipova (pomoć: ispiši sizeof(PlainOldClass) i sizeof(CoolClass)). Objasnite dobivenu razliku. Ako dobijete rezultate koje ne možete objasniti, pročitajte kada i zašto prevoditelj [**nadopunjava**](https://en.wikipedia.org/wiki/Data_structure_alignment#Data_structure_padding) objekte (engl. padding).

sizeof(PlainOldClass) = 4 🡪 pohranjuje samo jednu varijablu int (4B)

sizeof(CoolClass) = 8 🡪 varijabla int (4B) + pokazivač na vtable (4B)

**4. Vremenska cijena dinamičkog polimorfizma**

1. Pronađite dijelove assemblerskog kôda u kojima se odvija alociranje memorije za objekte poc i \*pb.

Trebalo bi se odviti alociranje memorije za objekt poc na stogu, ali nema toga u strojnom kodu. Razlog: kompajler radi neku optimizaciju (npr. sprema varijablu u registar ili zanemaruje varijablu jer nije inicijalizirana).

push ebp

mov ebp, esp

sub esp, 4 🡪 sizeof(PlainOldClass)

za \*pb :

mov DWORD PTR [esp], 8 push 8

call \_\_Znwj call operator new(unsigned int)

Konstanta 8 se stavlja na stog jer operator new (\_\_Znwj) kao parametar prima unsigned int (8B). Tamo se onda varijabla alocira na gomili.

2. Objasnite razliku u načinu alociranja tih objekata.

Memorija za poc (lokalna varijalba) se alocira na stogu, tijekom prevođenja. Memorija za \*pb (operator new) se alocira na gomili, tijekom izvođenja.

3. Pronađite dio assemblerskog kôda koji je zadužen za poziv konstruktora objekta poc, ako takav poziv postoji.

Ne postoji. PlainOldClass nema virtualnu tablicu i ne inicijalizira ništa.

4. Pronađite dio assemblerskog kôda koji je zadužen za poziv konstruktora objekta \*pb. Razmotrite kako se točno izvršava taj kôd. Što se u njemu događa?

call \_\_ZN9CoolClassC1Ev

call CoolClass::CoolClass() [complete object constructor]

Poziva se bazni konstruktor:

call \_\_ZN4BaseC2Ev

U memoriju objekta se upisuje referenca na pripadnu tablicu virtualnih funkcija:

mov edx, OFFSET FLAT:\_\_ZTV9CoolClass+8

mov eax, DWORD PTR [ebp-12]

mov DWORD PTR [eax], edx

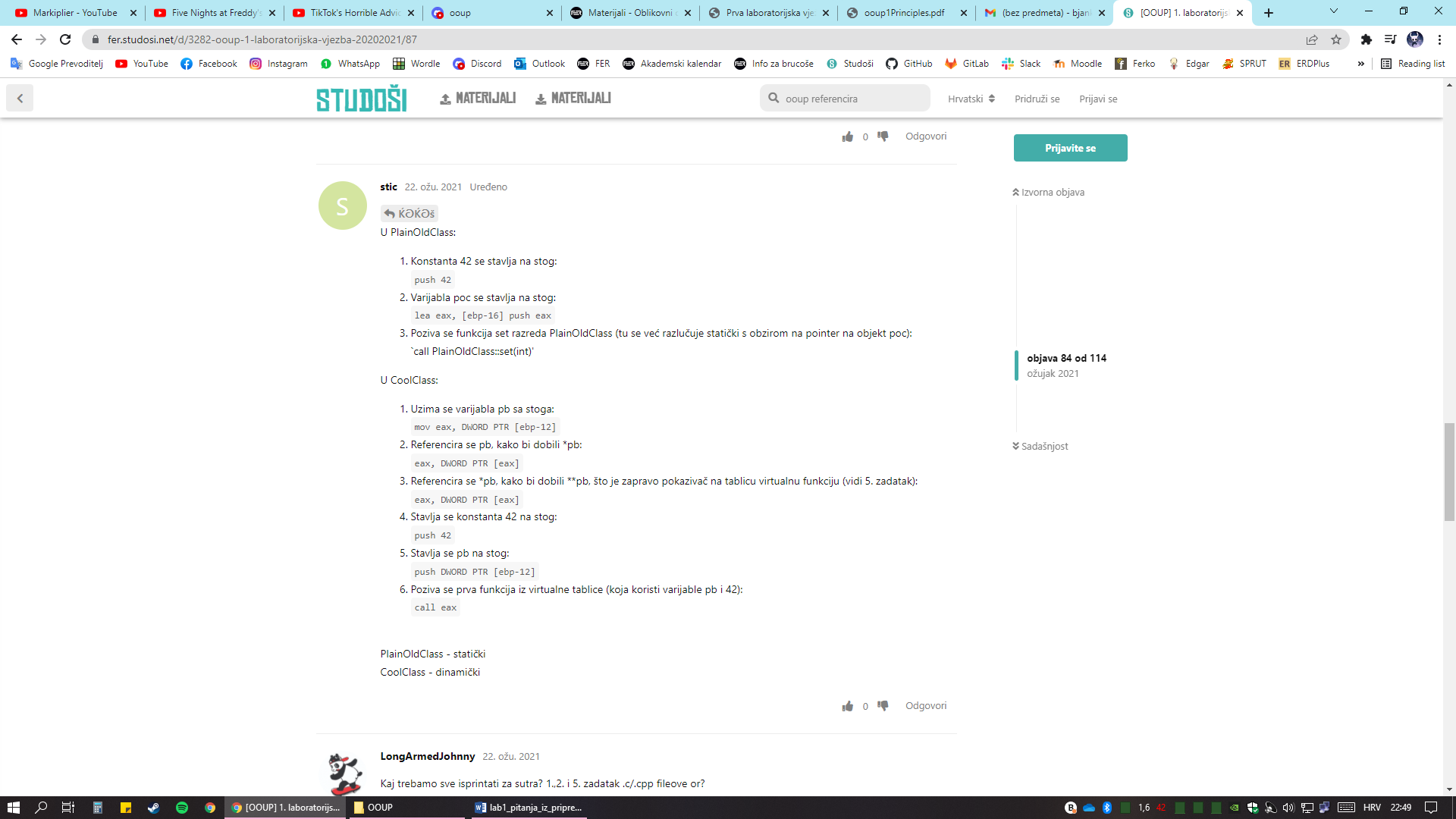
Ta referenca se nalazi odmah na prvom mjestu memorijske lokacije zauzete za instancu razreda CoolClass. Pokazivač na virtualnu tablicu je odmah na početku objekta jer ima istu memorijsku adresu kao i objekt klase.

(?) Inicijalizira varijable na defaultne vrijednosti. Varijabla tipa int nema defaultnu vrijednost pa biti inicijalizirana na neku random vrijednost.

5. Promotrite kako je prevoditelj izveo pozive pb->set i poc.set. Objasnite razliku između izvedbi tih dvaju poziva. Koji od ta dva poziva zahtijeva manje instrukcija? Za koju od te dvije izvedbe bi optimirajući prevoditelj mogao generirati kôd bez instrukcije CALL odnosno izravno umetnuti implementaciju funkcije (eng. inlining)?

poc.set 🡪 Konstanta 42 se stavlja na stog. Varijabla poc se stavlja na stog. Poziva se funkcija set razreda PlainOldClass.

pb->set 🡪 Uzima se varijabla pb. Dvaput se dereferencira pb kako bi dobili pokazivač na virtualnu tablicu. Konstanta 42 se stavlja na stog. Varijabla pb se stavlja na stog. Poziva se prva funkcija iz virtualne tablice (koja koristi varijable pb i 42).



Poziv poc.set zahtijeva manje instrukcija jer se radi o nevirtualnoj funkciji. Nevirtualne funkcije se direktno pozivaju s neke memorijske lokacije i nije potrebno dohvaćati i dereferencirati pokazivač na virtualnu tablicu. Inlining je moguć za nevirtualne funkcije.

6. Pronađite asemblerski kôd za definiciju i inicijalizaciju tablice virtualnih funkcija razreda CoolClass.

\_\_ZTV9CoolClass:

.long 0

.long \_\_ZTI9CoolClass

.long \_\_ZN9CoolClass3setEi

.long \_\_ZN9CoolClass3getEv

vtable for CoolClass:

.long 0

.long typeinfo for CoolClass

.long CoolClass::set(int)

.long CoolClass::get()

**6. Polimorfizam tijekom konstruiranja objekta**

Objasnite ispis programa analizirajući prevedeni strojni kod. Obratite pažnju na to tko, kada i gdje postavlja/modificira pokazivač na tablicu virtualnih funkcija.

Pri kreiranju objekta tipa Derived se poziva se konstruktor nadrazreda tj. razreda Base koji u svom konstruktoru poziva metodu metoda() koja zatim poziva metodu virtualnaMetoda() nad trenutnim objektom (a on je tipa Base, pa se izvede ispis da smo trenutno u baznoj implementaciji). Zatim se poziva konstruktor razreda Derived koji ponavlja gornje opisani postupak i pošto smo sad u razredu Derived, ispisuje se da smo u izvedenoj implementaciji. Na kraju se ponovno zove metoda metoda() u main programu preko objekta koji je tipa Derived i on naravno poziva virtualnu metodu koju sadrži te se opet nalazimo u izvedenoj implementaciji.